A whirlwind tour of scikit-learn

Andreas Müller
Columbia University,
scikit-learn

Alfred P. Sloan
FOUNDATION

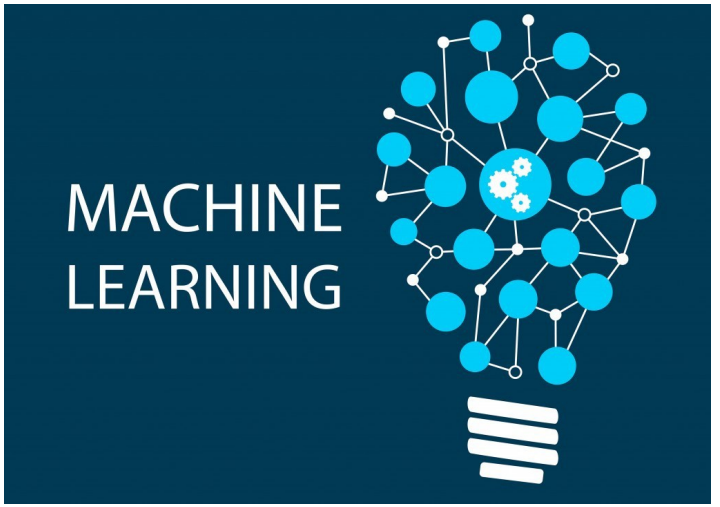
universitätbonn ais

NYU | Center for Data Science

COLUMBIA UNIVERSITY
Data Science Institute

What is scikit-learn?


python


MACHINE
LEARNING

Mission

Commoditize and Democratize Machine Learning

Classification
Regression
Clustering
Semi-Supervised Learning
Feature Selection
Feature Extraction
Manifold Learning
Dimensionality Reduction
Kernel Approximation
Hyperparameter Optimization
Evaluation Metrics
Out-of-core learning
……

okc

dataiku
Write your own data story

Spotify®

The New York Times

1000+ research papers


Alexandre Gramfort
agramfort


Alexander Fabisch
AlexanderFabisch


Alexandre Passos
alextp


Andreas Mueller
amueller


Arnaud Joly
arjoly


Brian Holt
bdholt1

bthirion
bthirion


Chris Filo Gorgolewski
chrisfilo


David Cournapeau
cournape


Duchesnay
duchesnay


David Warde-Farley
dwf


Fabian Pedregosa
fabianp


Gael Varoquaux
GaelVaroquaux


Gilles Louppe
glouppe


Jake Vanderplas
jakevdp


Jaques Grobler
jaquesgrobler


Jan Hendrik Metzen
jmetzen


Jacob Schreiber
jmschrei


Joel Nothman
jnothman


Kyle Kastner
kastnerkyle


Lars
larsmans


Loïc Estève
lesteve


Shiqiao Du
lucidfrontier45


Mathieu Blondel
mblondel


Manoj Kumar
MechCoder


Noel Dawe
ndawe


Nelle Varoquaux
NelleV


Olivier Grisel
ogrisel


Paolo Losi
paolo-losi


Peter Prettenhofer
pprett


(Venkat) Raghav (Rajagopalan)
raghavrv


Robert Layton
robertlayton


Ron Weiss
ronw


Satrajit Ghosh
satra


sklearn-ci


sklearn-wheels


Tom Dupré la Tour
TomDLT


Vlad Niculae
vene


Virgile Fritsch
VirgileFritsch


Vincent Michel
vmichel

Wei Li
weilinear

Yaroslav Halchenko
yarikoptic

Basic API

Representing Data

one sample

$$X = \begin{pmatrix} 1.1 & 2.2 & 3.4 & 5.6 & 1.0 \\ 6.7 & 0.5 & 0.4 & 2.6 & 1.6 \\ 2.4 & 9.3 & 7.3 & 6.4 & 2.8 \\ 1.5 & 0.0 & 4.3 & 8.3 & 3.4 \\ 0.5 & 3.5 & 8.1 & 3.6 & 4.6 \\ 5.1 & 9.7 & 3.5 & 7.9 & 5.1 \\ 3.7 & 7.8 & 2.6 & 3.2 & 6.3 \end{pmatrix}$$

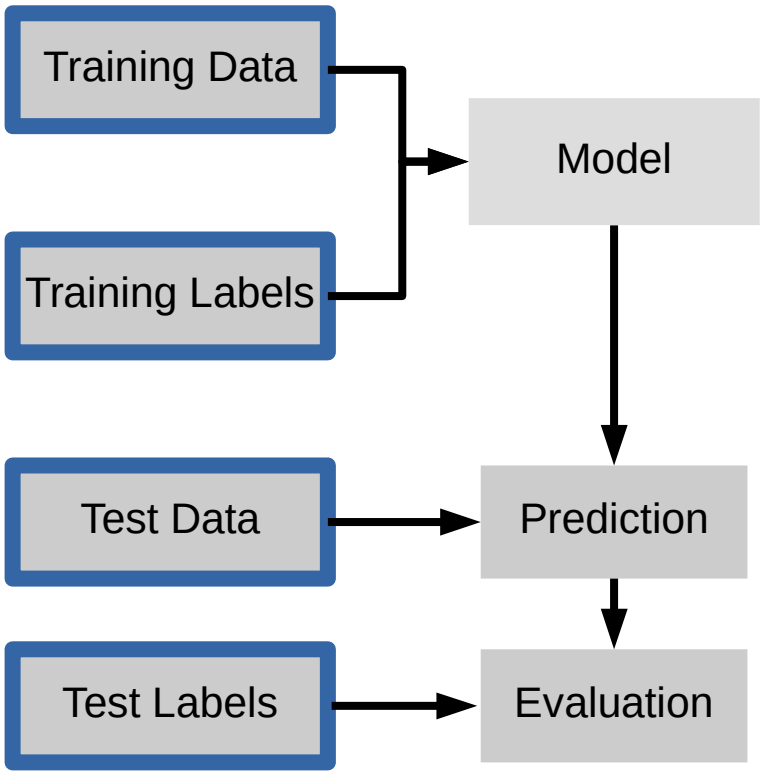
one feature

$$y = \begin{pmatrix} 1.6 \\ 2.7 \\ 4.4 \\ 0.5 \\ 0.2 \\ 5.6 \\ 6.7 \end{pmatrix}$$

outputs / labels

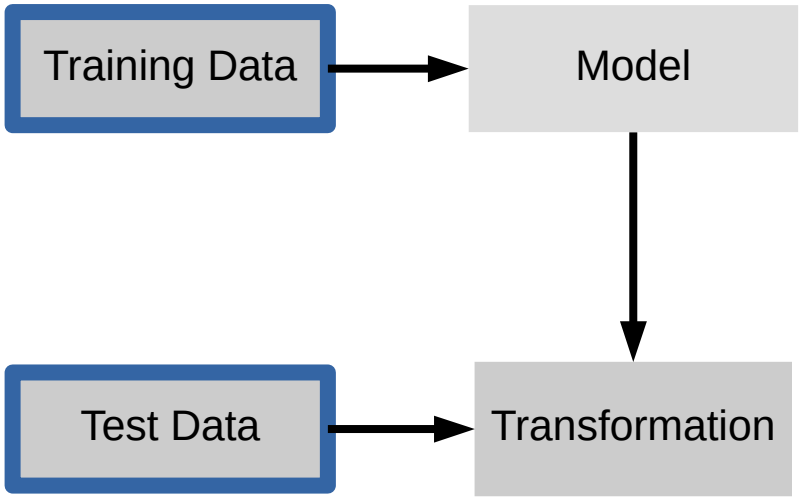
```
clf = RandomForestClassifier()

clf.fit(X_train, y_train)
```

```
y_pred = clf.predict(X_test)

clf.score(X_test, y_test)
```



Unsupervised Transformations

```
pca = PCA()

pca.fit(X_train)

X_new = pca.transform(X_test)
```

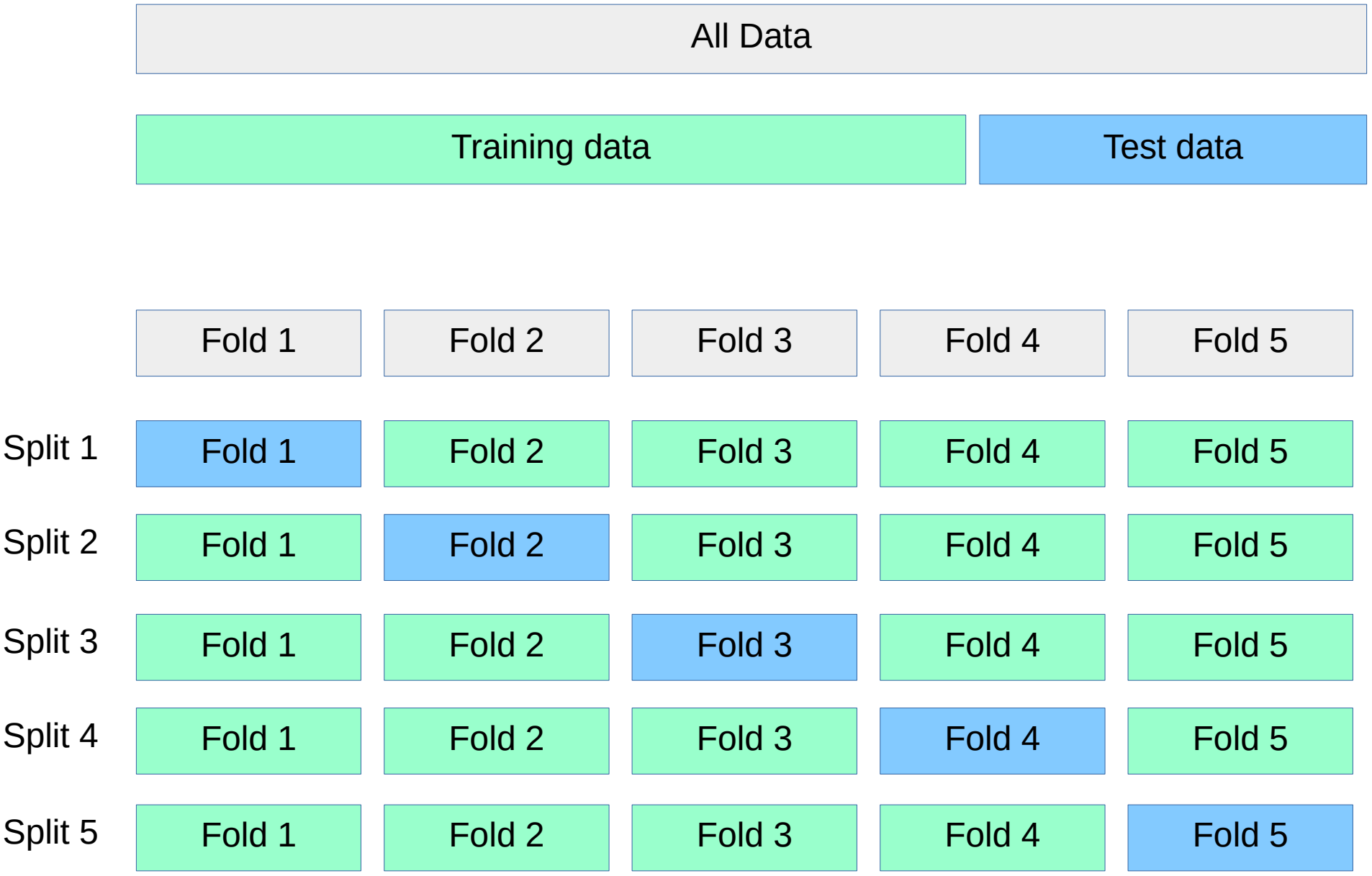


Core API Summary

`estimator.fit(X, [y])`

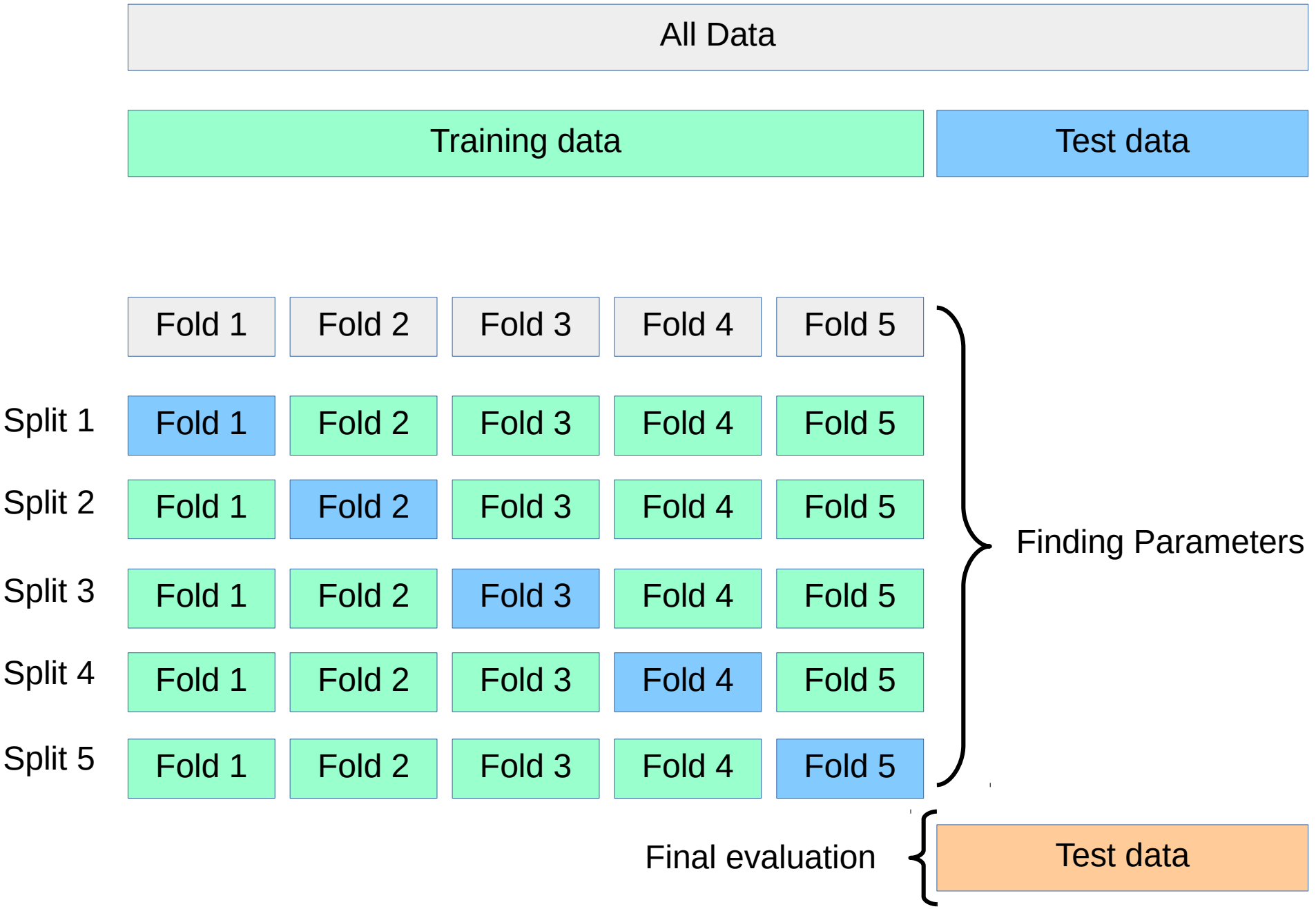
`estimator.predict`	**`estimator.transform`**
Classification	Preprocessing
Regression	Dimensionality reduction
Clustering	Feature selection
	Feature extraction

Model Evaluation and Model Selection


All Data
Training data
Test data
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Split 1
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Split 2
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Split 3
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Split 4
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Split 5
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

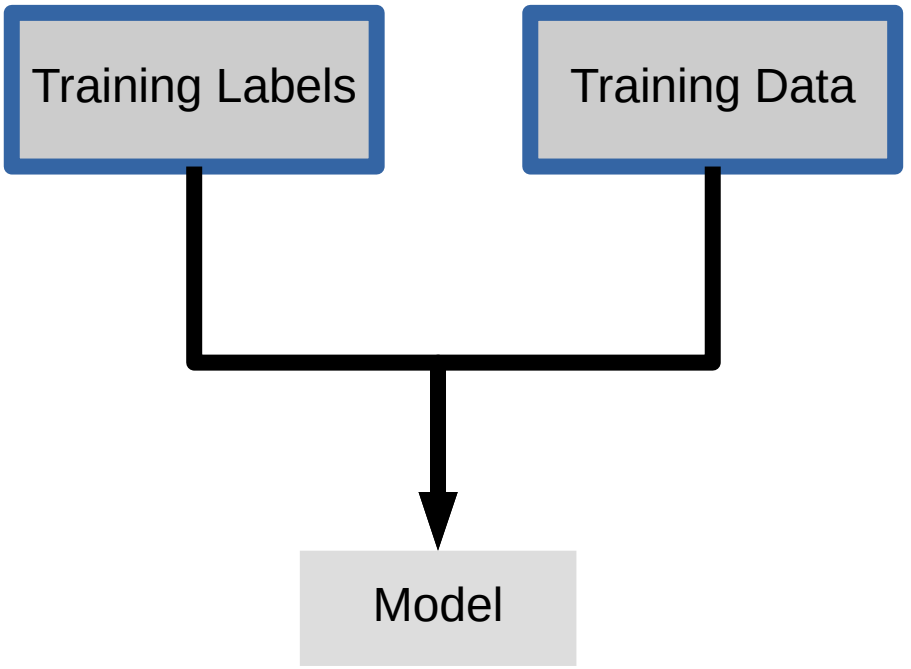
Cross-Validation

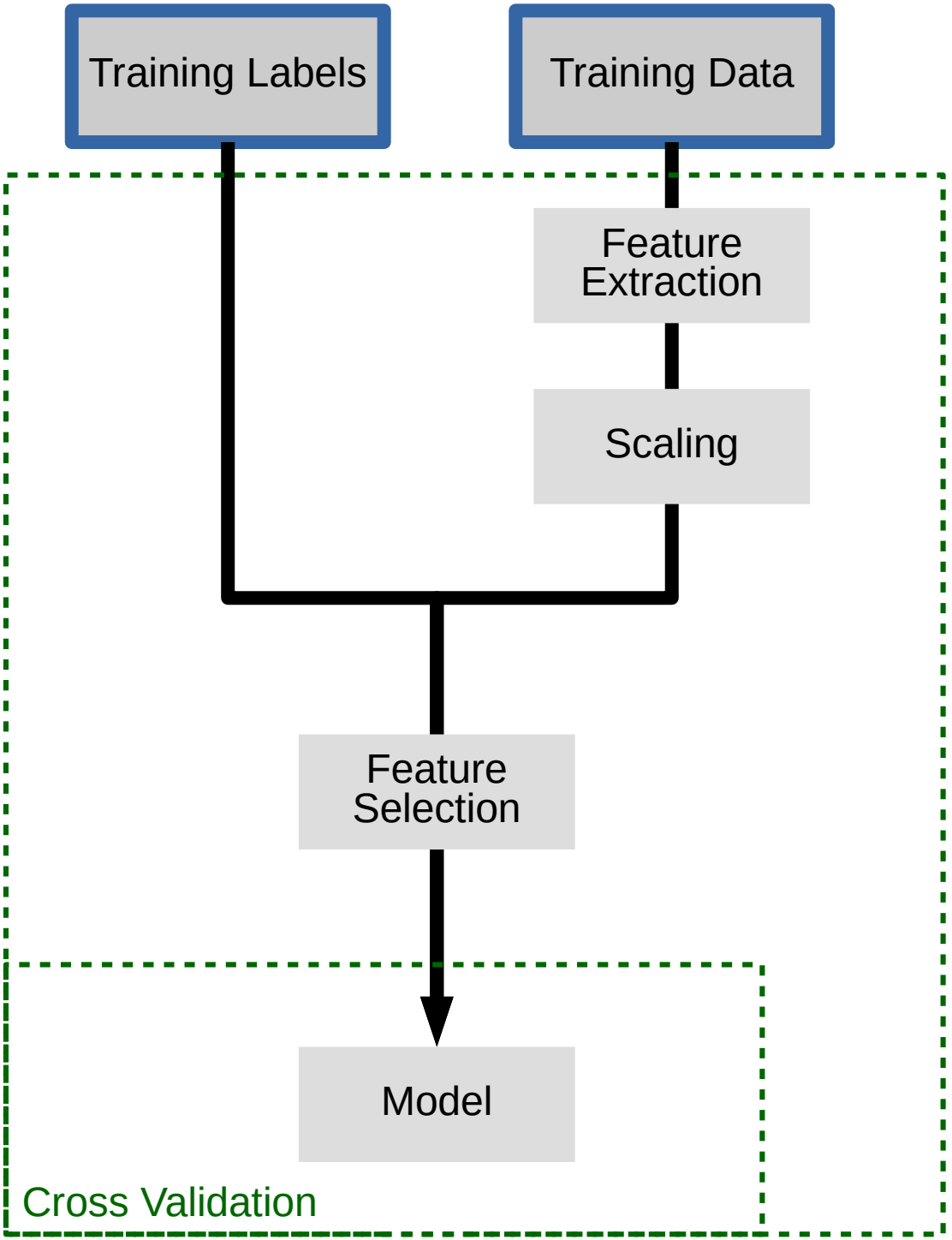
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(SVC(), X, y, cv=5)
print(scores)
>> [ 0.92  1.    1.    1.    1.  ]
```


All Data
Training data
Test data
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Split 1
Split 2
Split 3
Split 4
Split 5
Finding Parameters
Final evaluation
Test data

Cross -Validated Grid Search

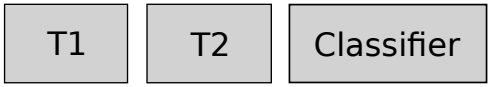
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y)

param_grid = {'C': 10. ** np.arange(-3, 3),
              'gamma': 10. ** np.arange(-3, 3)}
grid = GridSearchCV(SVC(), param_grid=param_grid)
grid.fit(X_train, y_train)
grid.predict(X_test)
grid.score(X_test, y_test)
```


Training Labels
Training Data
Model

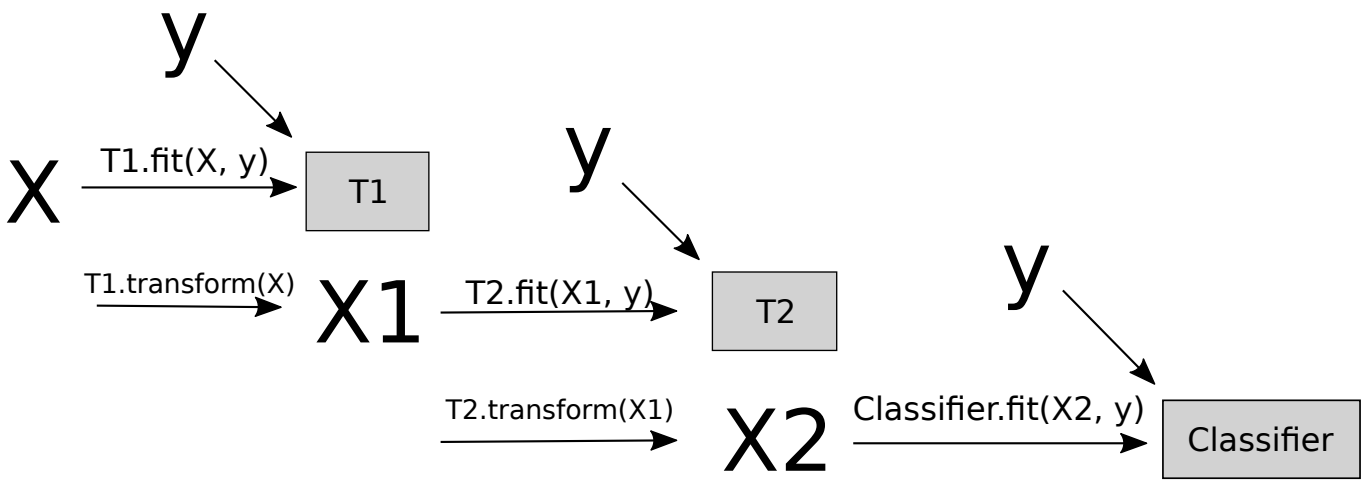

Training Labels
Training Data
Feature Extraction
Scaling
Feature Selection
Model
Cross Validation

Pipelines

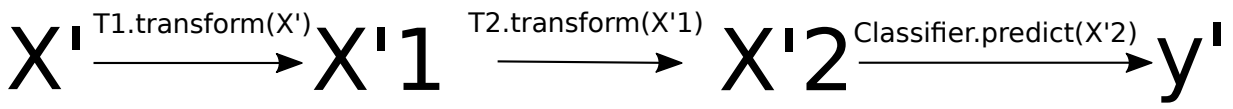
```
pipe = make_pipeline(T1(), T2(), Classifier())
```




```
pipe.fit(X, y)
```




```
pipe.predict(X')
```



Pipelines

```
from sklearn.pipeline import make_pipeline

pipe = make_pipeline(StandardScaler(), SVC())
pipe.fit(X_train, y_train)
pipe.predict(X_test)
```

```
param_grid = {'svc__C': 10. ** np.arange(-3, 3),
              'svc__gamma': 10. ** np.arange(-3, 3)}

scaler_pipe = make_pipeline(StandardScaler(), SVC())
grid = GridSearchCV(scaler_pipe, param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
```

Combining Pipelines and Grid Search

Proper cross-validation

```
param_grid = {'svc__C': 10. ** np.arange(-3, 3),
              'svc__gamma': 10. ** np.arange(-3, 3)}

scaler_pipe = make_pipeline(StandardScaler(), SVC())
grid = GridSearchCV(scaler_pipe, param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
```

Combining Pipelines and Grid Search II

Searching over parameters of the preprocessing step

```
param_grid = {'selectkbest__k': [1, 2, 3, 4],
              'svc__C': 10. ** np.arange(-3, 3),
              'svc__gamma': 10. ** np.arange(-3, 3)}

scaler_pipe = make_pipeline(SelectKBest(), SVC())
grid = GridSearchCV(scaler_pipe, param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
```

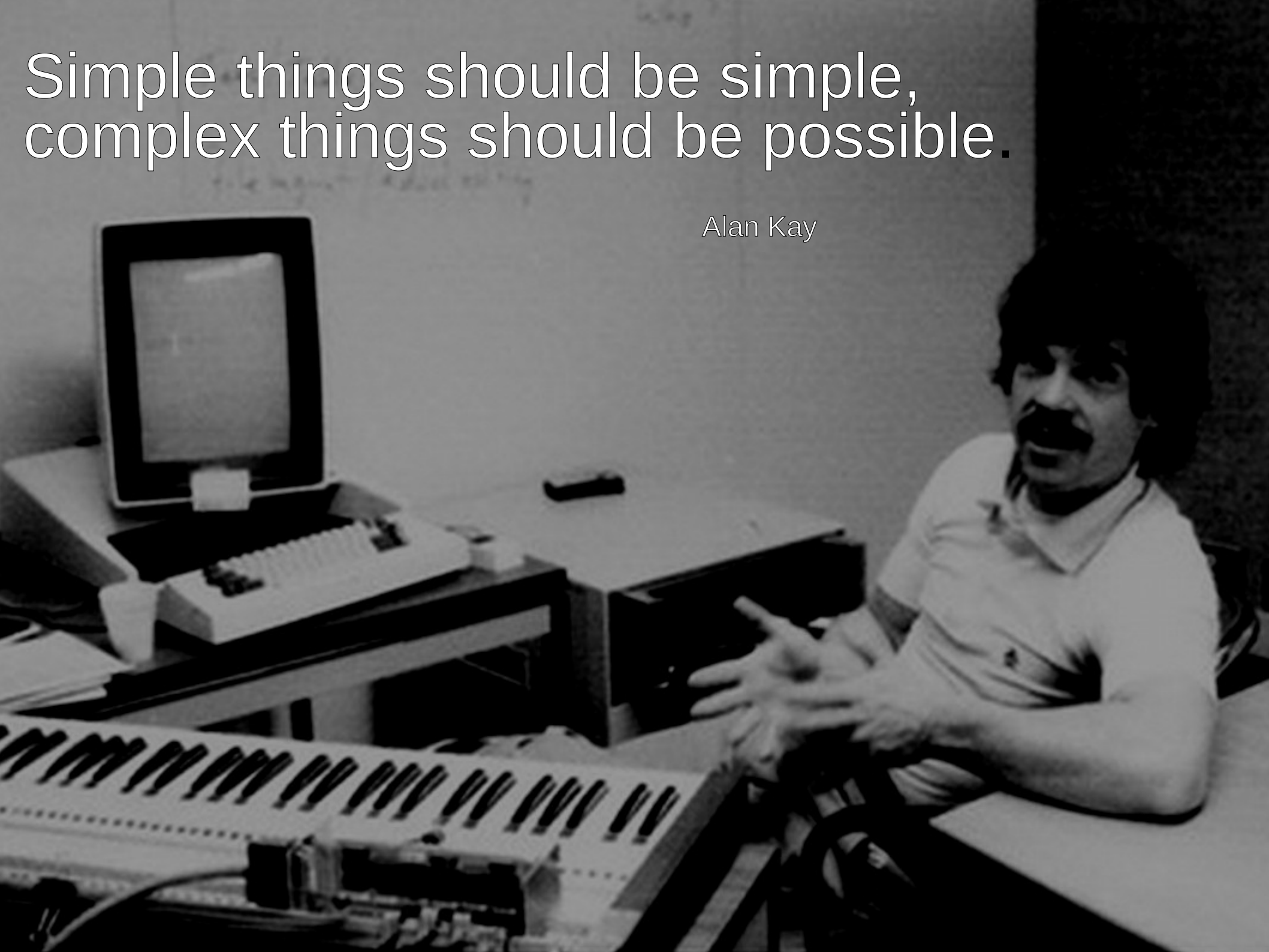
More API

Scorers

```
grid = GridSearchCV(SVC(), params, scoring='roc_auc')

def my_acc_scorer(est, X, y):
    y_pred = est.predict(X)
    return (y == y_pred).mean()

grid = GridSearchCV(SVC(), params,
                                scoring=my_acc_scorer)
```

Cross-validation iterators

```
grid = GridSearchCV(SVC(), param_grid, cv=5)

cv = KFold(n_split=5, shuffle=True, random_state=3)

grid = GridSearchCV(SVC(), param_grid, cv=cv)


cv = RepeatedKFold(n_split=5, n_repeats=10)
grid = GridSearchCV(SVC(), param_grid, cv=cv)
```


Simple things should be simple,
complex things should be possible.
Alan Kay

Simplicity

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
lr.score(X_test, y_test)
```

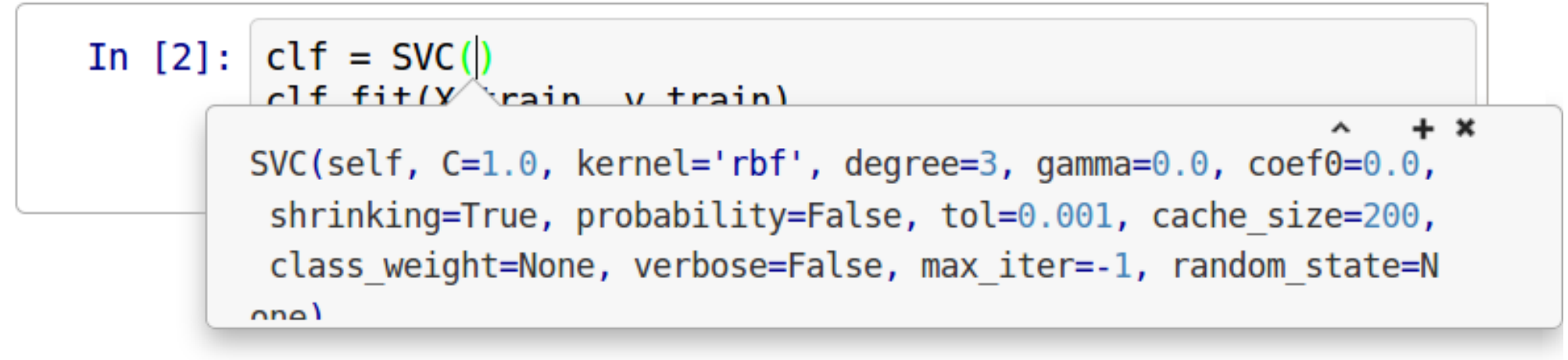
Consistency

```
grid = GridSearchCV(svm,param_grid)
grid.fit(X_train, y_train)
grid.score(X_test, y_test)
```

Composition

```
feature_selection = RFECV(NaiveBayes())
pipe = make_pipeline(feature_selection,
                     RandomForestClassifier())
grid = GridSearchCV(pipe, param_grid)
```

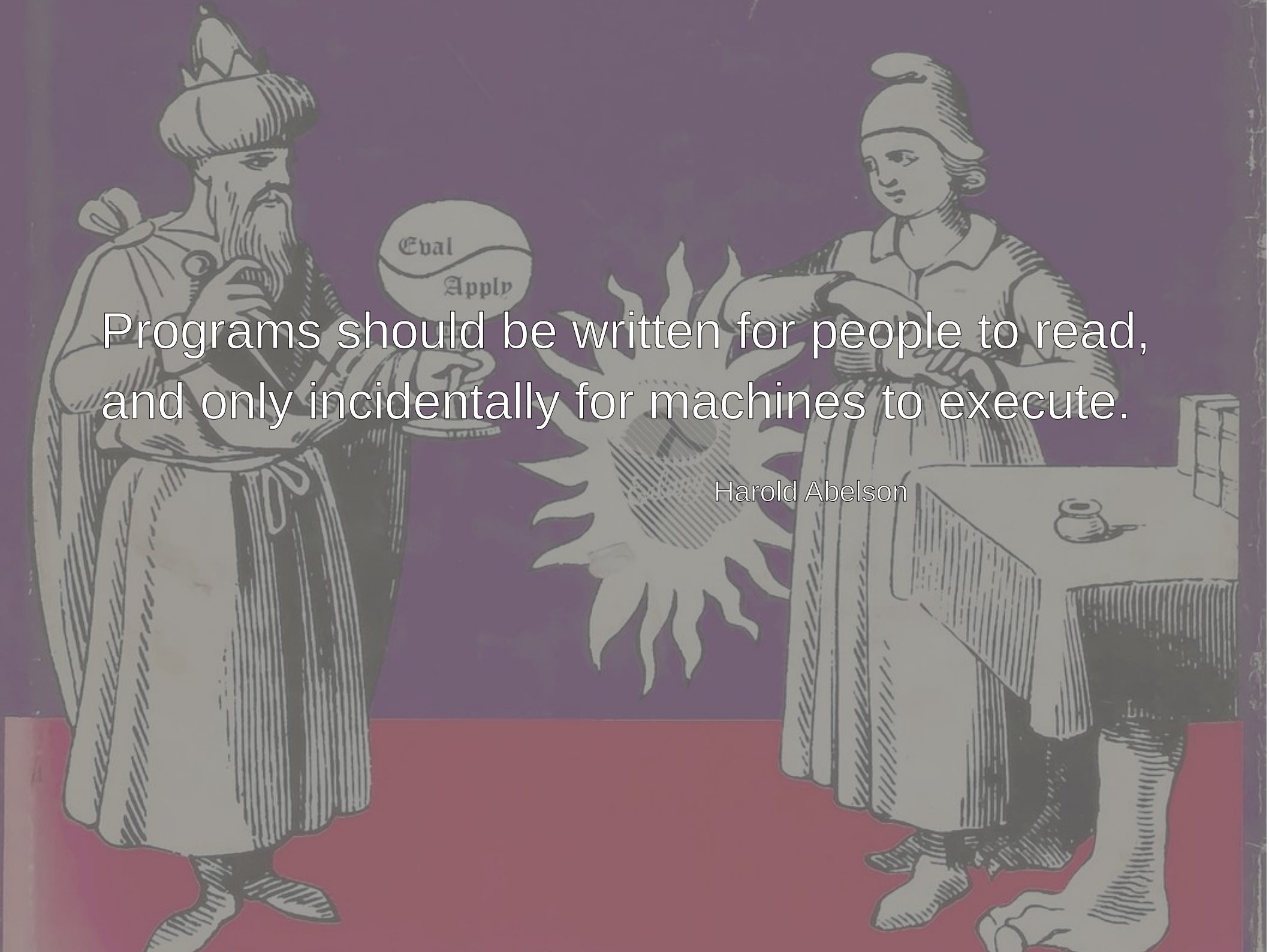
Default Parameters



Flat Class Hierarchy, Few Types

- Numpy arrays / sparse matrices
- Estimators
- Cross-validation objects
- Scorers

Development Practices



Programs should be written for people to read, and only incidentally for machines to execute.

Harold Abelson

Standards for OSS

Everything discussed in the open.
Every convention and process documented.

Development guide

http://scikit-learn.org/dev/developers/contributing.html

Contains:

- API details
- Bug report guidelines
- PR guidelines
- Reviewing guidelines
- How to find issues
- Details of CI

Even more at http://scikit-learn.org/dev/developers/

Deprecations / backward compatibility

- Don’t change any behavior (except bug fixes)

```
@property
@deprecated("Attribute labels_ was deprecated in version 0.13 and "
            "will be removed in 0.15. Use 'classes_' instead")
def labels_(self):
    return self.classes_
```

check_estimator

```python
class TemplateClassifier(BaseEstimator, ClassifierMixin):

    def __init__(self, demo_param='demo'):
        self.demo_param = demo_param

    def fit(self, X, y):

        # Check that X and y have correct shape
        X, y = check_X_y(X, y)
        # Store the classes seen during fit
        self.classes_ = unique_labels(y)

        self.X_ = X
        self.y_ = y
        # Return the classifier
        return self

    def predict(self, X):

        closest = np.argmin(euclidean_distances(X, self.X_), axis=1)
        return self.y_[closest]
```

```
check_estimator(TemplateClassifier)
```

```
AssertionError: Error message does not include the expected string: 'fit'.
Observed error message: "'TemplateClassifier' object has no attribute 'X_'"
```

Coming up for 0.20
(July / August 2018)

ColumnTransformer

```
# We create the preprocessing pipelines for both numeric and categorical data.
numeric_transformer = make_pipeline(SimpleImputer(), StandardScaler())
categorical_transformer = CategoricalEncoder('onehot-dense',
                                              handle_unknown='ignore')

preprocessing_pl = make_column_transformer(
    (numeric_features, numeric_transformer),
    (categorical_features, categorical_transformer),
    remainder='drop'
)

# Append classifier to preprocessing pipeline.
# Now we have a full prediction pipeline.
clf = make_pipeline(preprocessing_pl, LogisticRegression())
```

OneHotEncoder for Strings

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> enc = OneHotEncoder(handle_unknown='ignore')
>>> X = [['Male', 1], ['Female', 3], ['Female', 2]]
>>> enc.fit(X)
...
OneHotEncoder(categorical_features=None, categories=None,
       dtype=<... 'numpy.float64'>, handle_unknown='ignore',
       n_values=None, sparse=True)
```

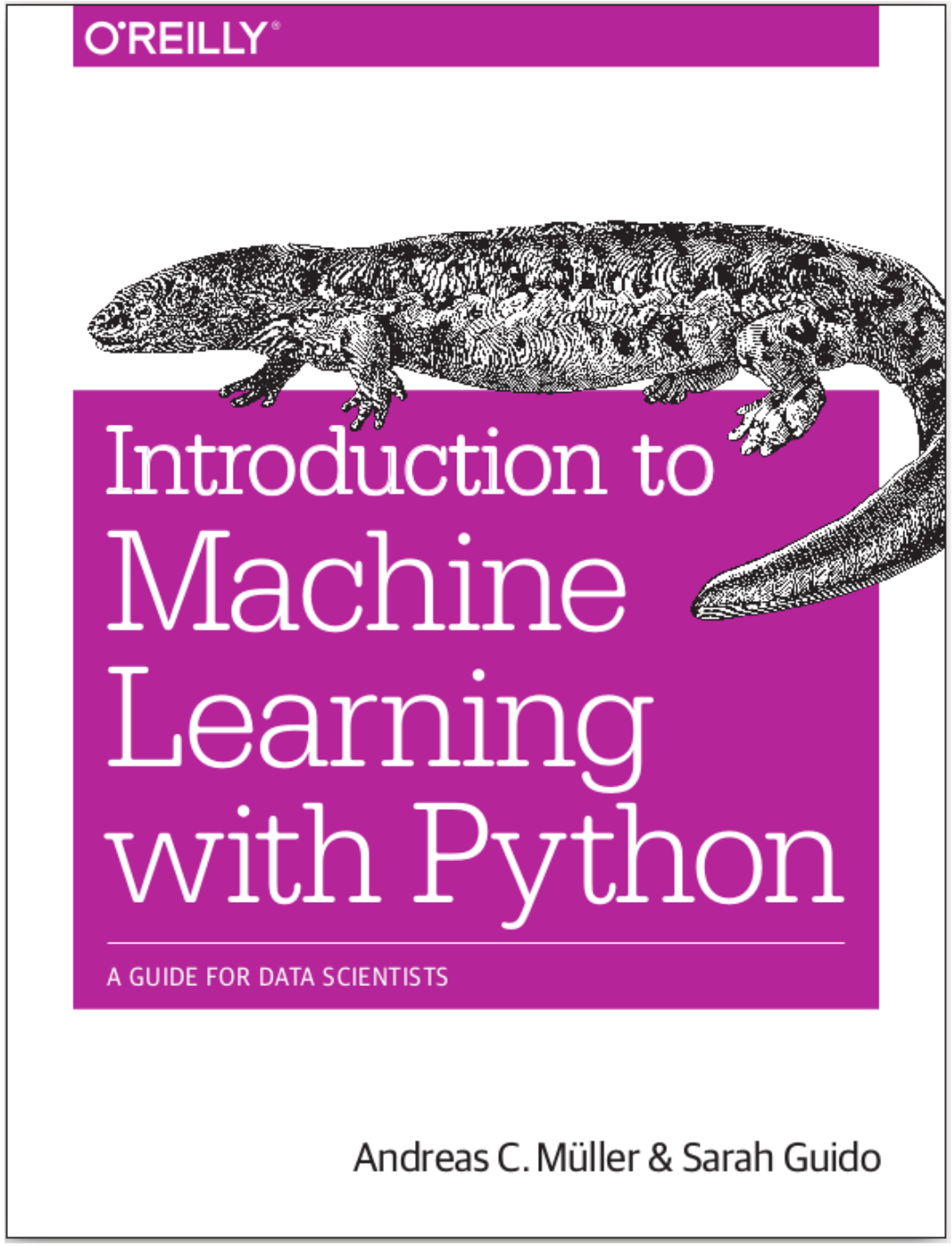
```
>>> enc.categories_
[array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
>>> enc.transform([['Female', 1], ['Male', 4]]).toarray()
array([[1., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0.]])
>>> enc.inverse_transform([[0, 1, 1, 0, 0], [0, 0, 0, 1, 0]])
array([['Male', 1],
       [None, 2]], dtype=object)
```

NaN handling in Scalers
Better imputation

(my personal) Roadmap

Make simple things simple again!

Pandas Integration
Feature Names
More convenient preprocessing
Plotting Tools
Estimator Tags



amueller.github.io

@amuellerml

@amueller

importamueller
@gmail.com